

# Applied Computer Algebra: Using FeynArts and FormCalc

Thomas Hahn

Max-Planck-Institut für Physik  
München

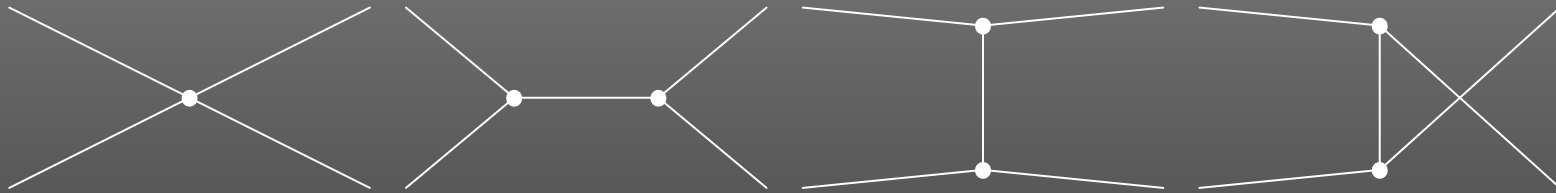
<http://wwwth.mpp.mpg.de/members/hahn/fafc.pdf>

<http://wwwth.mpp.mpg.de/members/hahn/fafc.tar.gz>



# Feynman Diagram Cookbook

1. Draw all possible types of diagrams with the given number of loops and external legs



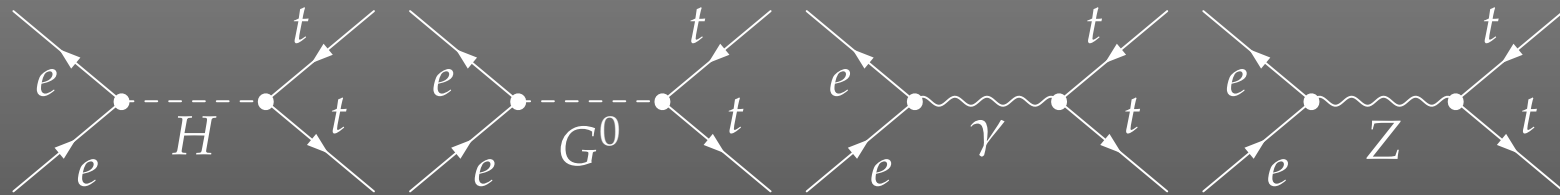
**Topological task, no physics input needed\***

\* Well, almost: need to know allowed adjacencies in physics model, e.g. renormalizable theories have at most 3- and 4-point vertices.



# Feynman Diagram Cookbook

## 2. Figure out what particles can run on each type of diagram



## Combinatorial task, requires physics input (model)

In this case, in the SM, three of the topologies were not realized though one was realized multiply.

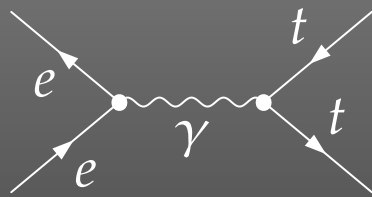
Note further that the e-e-scalar couplings are suppressed by  $m_e^2/M_W^2$  and thus usually neglected.

These are selections one would typically make at this stage, i.e. diagrammatically.



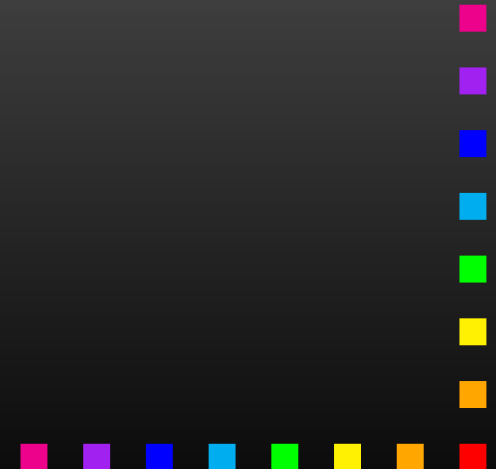
# Feynman Diagram Cookbook

## 3. Translate the diagrams into formulas by applying the Feynman rules



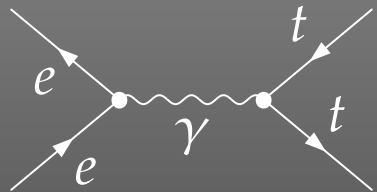
$$= \underbrace{\langle v_1 | i e \gamma^\mu | u_2 \rangle}_{\text{left vertex}} \underbrace{\frac{g_{\mu\nu}}{(k_1 + k_2)^2}}_{\text{propagator}} \underbrace{\langle u_4 | \left( -\frac{2}{3} i e \gamma^\nu \right) | v_3 \rangle}_{\text{right vertex}}$$

## Database look-up



# Feynman Diagram Cookbook

## 4. Contract the indices, take the traces, etc.



The diagram shows an incoming electron (e) and positron (e) on the left, meeting at a vertex. A wavy line representing a photon (γ) connects this vertex to another vertex on the right. From the right vertex, an outgoing top quark (t) and anti-top quark (t) emerge.

$$= \frac{8\pi\alpha}{3s} F_1, \quad F_1 = \langle v_1 | \gamma_\mu | u_2 \rangle \langle u_4 | \gamma^\mu | v_3 \rangle$$

Also, compute the fermionic matrix elements, e.g. by squaring and taking the trace:

$$\begin{aligned} |F_1|^2 &= \text{Tr} \{ (\not{k}_1 - m_e) \gamma_\mu (\not{k}_2 + m_e) \gamma_\nu \} \text{Tr} \{ (\not{k}_4 + m_t) \gamma^\mu (\not{k}_3 - m_t) \gamma^\nu \} \\ &= \frac{1}{2} s^2 + st + (m_e^2 + m_t^2 - t)^2 \end{aligned}$$

## Algebraic simplification

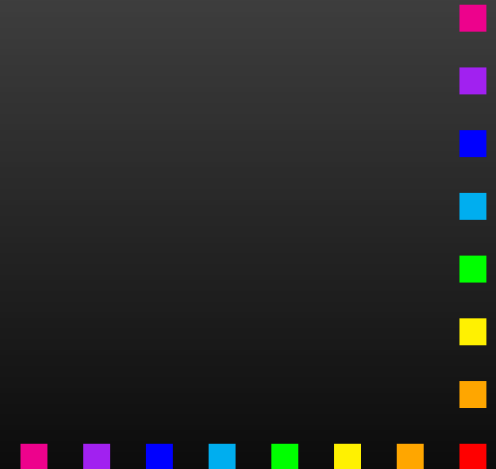
# Feynman Diagram Cookbook

5. Write the results up as a ..... program  
(put favorite language here)

5a. Debug that program

6. Run it to produce numerical values

## Programming



# Recipe for Feynman Diagrams



Thanks to  and  (and many others) we have a **Recipe for an ARBITRARY Feynman diagram up to one loop**

①	<b>Draw all possible types of diagrams</b>	<b>topological task</b>
②	<b>Figure out what particles can run on each type of diagram</b>	<b>combinatorical task</b>
③	<b>Translate the diagrams into formulas by applying the Feynman rules</b>	<b>database look-up</b>
④	<b>Contract the indices, take the traces, etc.</b>	<b>algebraic simplification</b>
⑤	<b>Write up the results as a computer program</b>	<b>programming</b>
⑥	<b>Run the program to get numerical results</b>	<b>waiting</b>



# Programming Techniques

- Very different tasks at hand.
- Some objects must/should be handled symbolically, e.g. tensorial objects, Dirac traces, dimension ( $D$  vs. 4).
- Reliable results required even in the presence of large cancellations.
- Fast evaluation desirable (e.g. for Monte Carlos).

## Hybrid Programming Techniques necessary

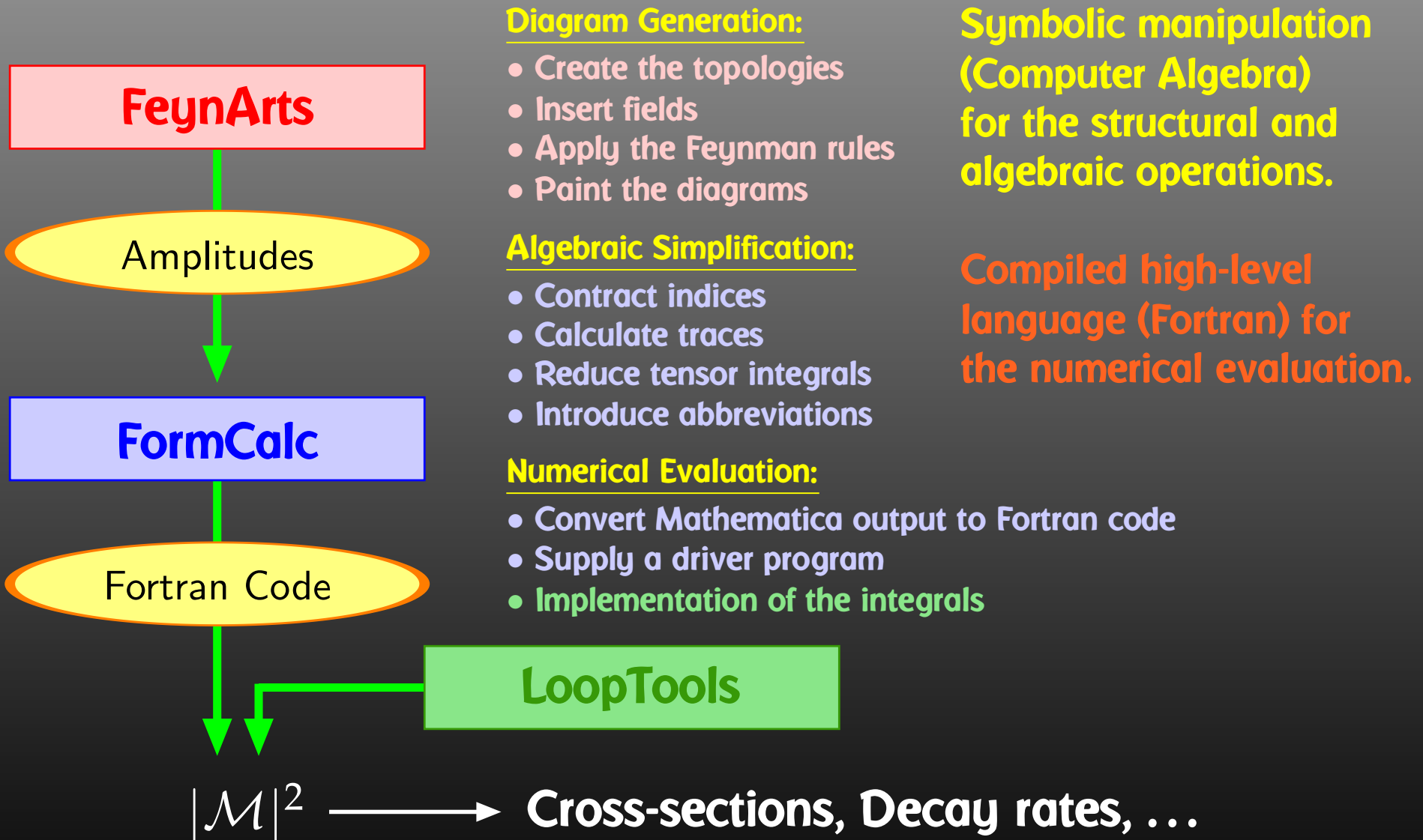
Symbolic manipulation (a.k.a. Computer Algebra) for the structural and algebraic operations.

Compiled high-level language (e.g. Fortran) for the numerical evaluation.





# Automated Diagram Evaluation



## One-loop since mid-1990s

Automated NLO computations is an industry today, with many packages becoming available in the last few years:

- GoSam, HELAC-NLO, aMC@NLO, MadLoop, OpenLoops, BlackHat, Rocket, ...

Here: **FeynArts (1991) + FormCalc (1995)**

FormCalc was doing largely the same as FeynCalc (1992) but used FORM for the time-consuming tasks, hence the name FormCalc.

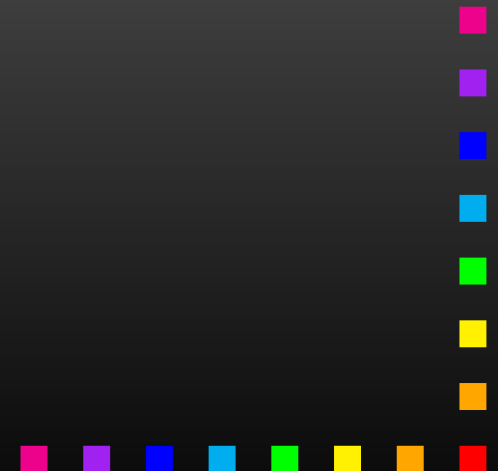
- Feynman-diagrammatic method,
- Analytic calculation as far as possible (any model),
- Generation of code for the numerical evaluation of the squared matrix element.



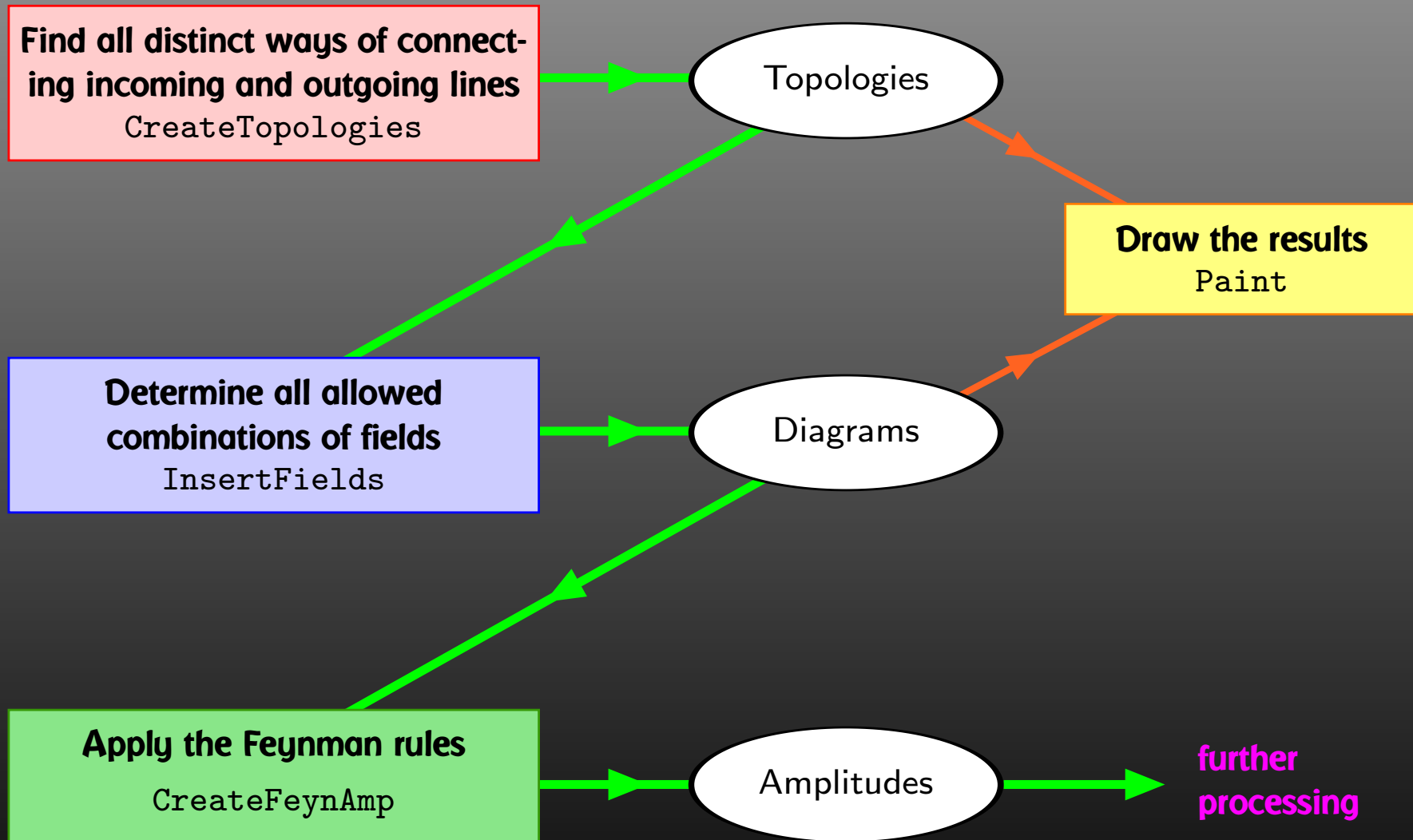
# Plan

Walk through the general setup of these programs and show some perhaps non-standard applications.

- 'Standard Candle' –  $e^+e^- \rightarrow t\bar{t}$ ,
- Resumming a coupling –  $\Delta_b$ ,
- Example from flavour physics –  $\Delta M_s$ .

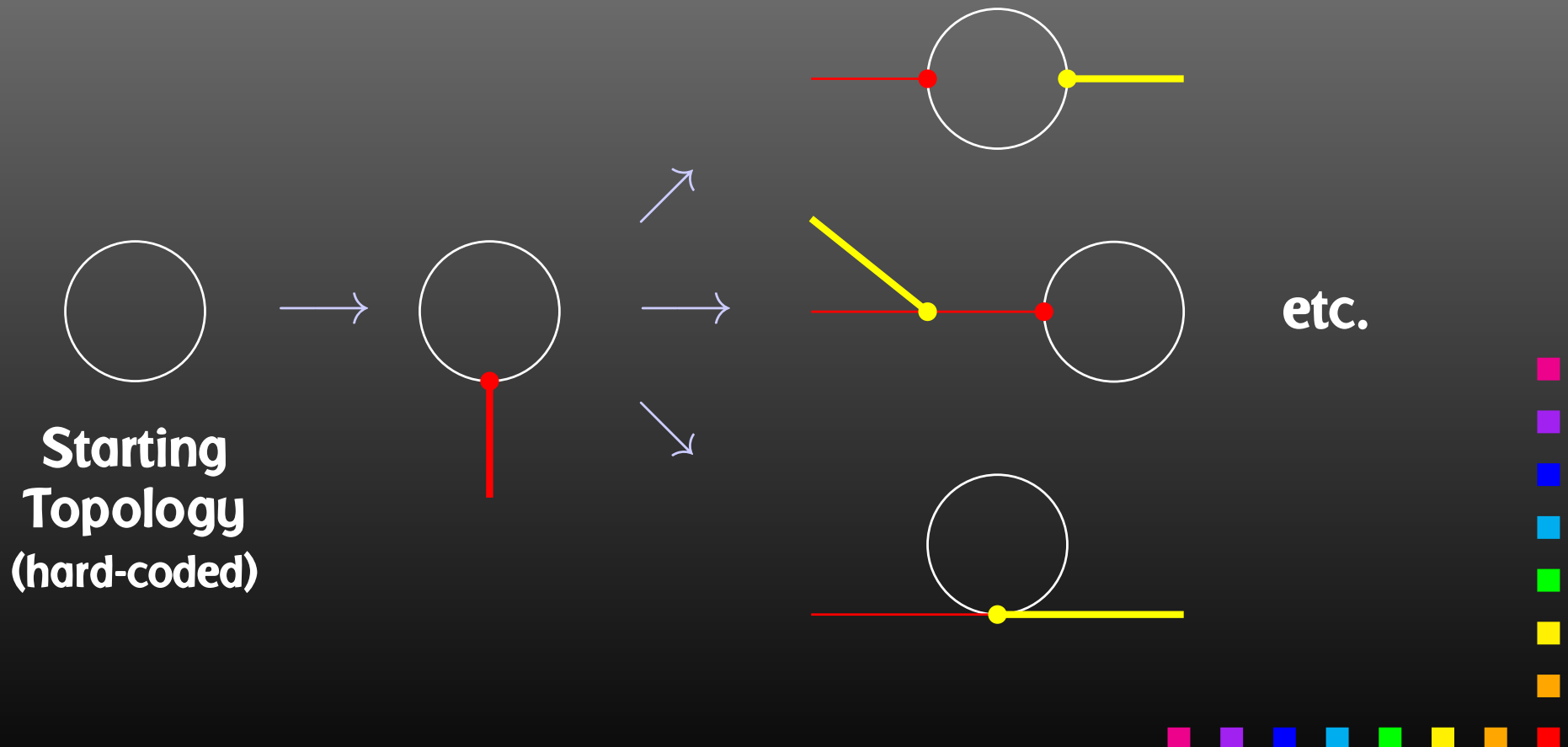


# FeynArts



# CreateTopologies

Algorithm: Start from **Zero-leg Topologies** and successively **add external legs**. This is not entirely self-sufficient, but few people would even want to use FeynArts beyond three loops.



# Three Levels of Fields

**Generic level, e.g.  $F, F, S$**

$$C(F_1, F_2, S) = G_L \mathbb{P}_L + G_R \mathbb{P}_R \quad \mathbb{P}_{R,L} = (\mathbb{1} \pm \gamma_5)/2$$

**Kinematical structure completely fixed, most algebraic simplifications (e.g. tensor reduction) can be carried out.**

**Classes level, e.g.  $-F[2], F[1], S[3]$**

$$\bar{\ell}_i \nu_j G : \quad G_L = -\frac{i e m_{\ell,i}}{\sqrt{2} \sin \theta_w M_W} \delta_{ij}, \quad G_R = 0$$

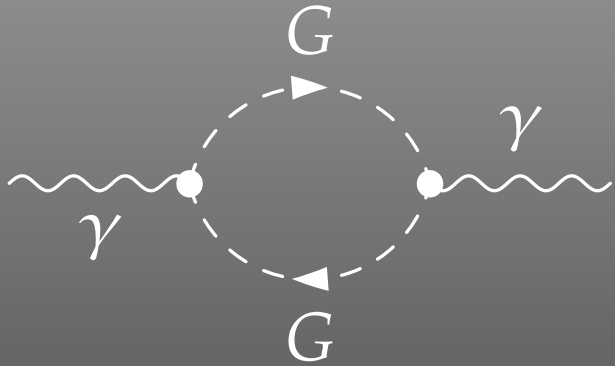
**Coupling fixed except for  $i, j$  (can be summed in do-loop).**

**Particles level, e.g.  $-F[2, \{1\}], F[1, \{1\}], S[3]$**

**insert fermion generation (1, 2, 3) for  $i$  and  $j$**

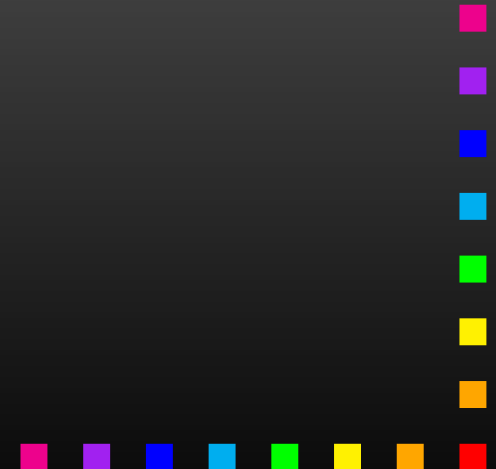


# Sample CreateFeynAmp output

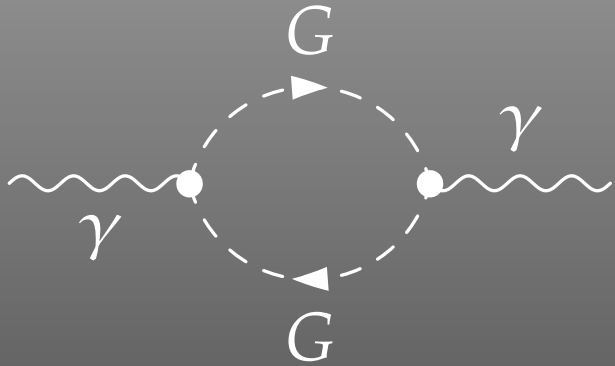


= FeynAmp[ identifier,  
*loop momenta*,  
*generic amplitude*,  
*insertions* ]

GraphID[Topology == 1, Generic == 1]

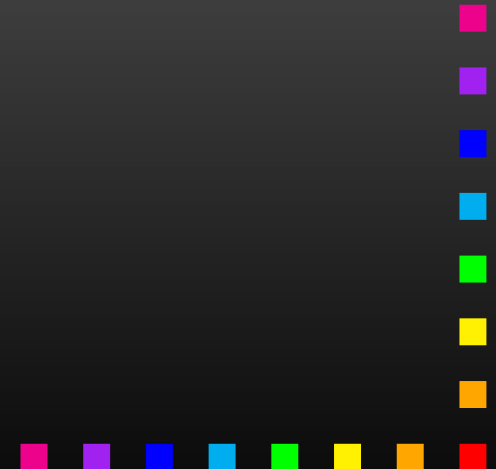


# Sample CreateFeynAmp output



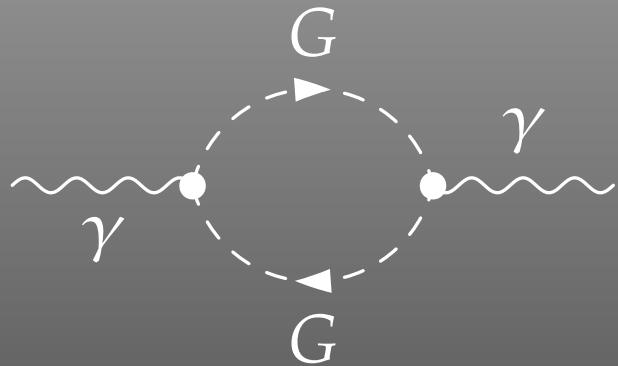
```
= FeynAmp[ identifier ,  
            loop momenta ,  
            generic amplitude ,  
            insertions ]
```

```
Integral[q1]
```





# Sample CreateFeynAmp output



= FeynAmp [ *identifier* ,  
*loop momenta* ,  
*generic amplitude* ,  
*insertions* ]

$\frac{1}{32 \text{ Pi}^4}$  RelativeCF .....prefactor

FeynAmpDenominator [  $\frac{1}{q1^2 - \text{Mass}[S[\text{Gen3}]]^2}$  ,  
 $\frac{1}{(-p1 + q1)^2 - \text{Mass}[S[\text{Gen4}]]^2}$  ] ..... loop denominators

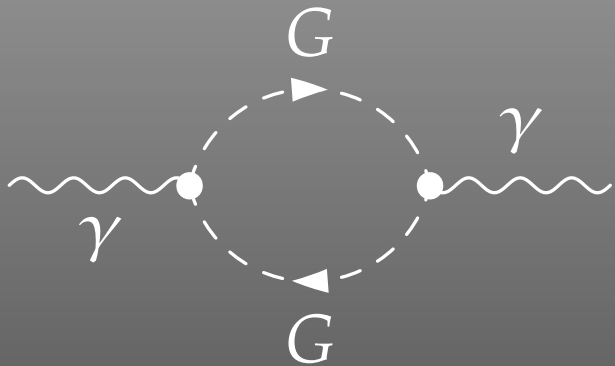
$(p1 - 2q1)[\text{Lor1}] (-p1 + 2q1)[\text{Lor2}]$  ..... kin. coupling structure

$\text{ep}[V[1], p1, \text{Lor1}] \text{ep}^*[V[1], k1, \text{Lor2}]$  ..... polarization vectors

$G_{\text{SSV}}^{(0)}[(\text{Mom}[1] - \text{Mom}[2])[\text{KI1}[3]]]$   
 $G_{\text{SSV}}^{(0)}[(\text{Mom}[1] - \text{Mom}[2])[\text{KI1}[3]]]$  , ..... coupling constants

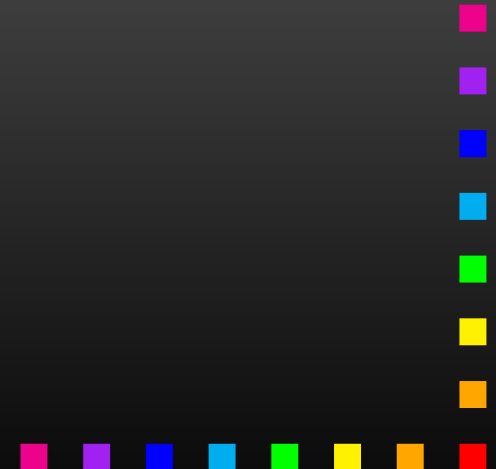


# Sample CreateFeynAmp output



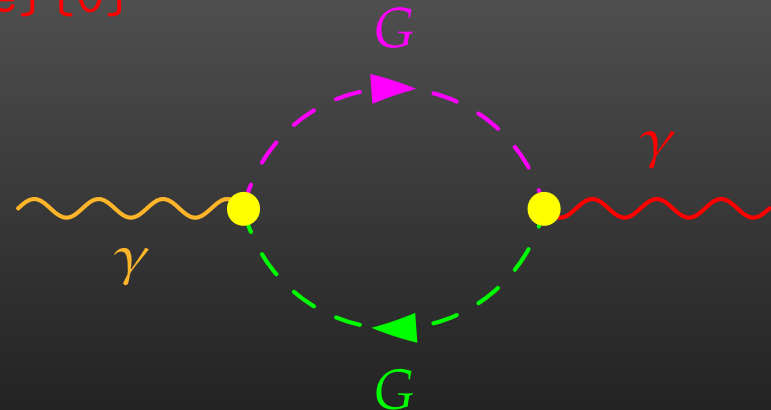
= FeynAmp[ *identifier*,  
*loop momenta*,  
*generic amplitude*,  
*insertions* ]

```
{ Mass[S[Gen3]],
  Mass[S[Gen4]],
  GSSV(0)[(Mom[1] - Mom[2])[KI1[3]]],
  GSSV(0)[(Mom[1] - Mom[2])[KI1[3]]],
  RelativeCF } ->
Insertions[Classes][{MW, MW, I EL, -I EL, 2}]
```



# Sample Paint output

```
\begin{feynartspicture}(150,150)(1,1)
\FADiagram{}
\FAProp(6.,10.)(14.,10.)(0.8,){ScalarDash}{-1}
\FALabel(10.,5.73)[t]{G}
\FAProp(6.,10.)(14.,10.)(-0.8,){ScalarDash}{1}
\FALabel(10.,14.27)[b]{G}
\FAProp(0.,10.)(6.,10.)(0.,){Sine}{0}
\FALabel(3.,8.93)[t]{\gamma}
\FAProp(20.,10.)(14.,10.)(0.,){Sine}{0}
\FALabel(17.,11.07)[b]{\gamma}
\FAVert(6.,10.){0}
\FAVert(14.,10.){0}
\end{feynartspicture}
```



**Technically: uses its own PostScript prologue.**

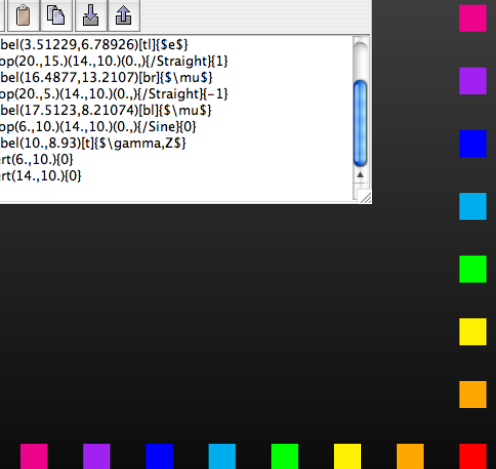
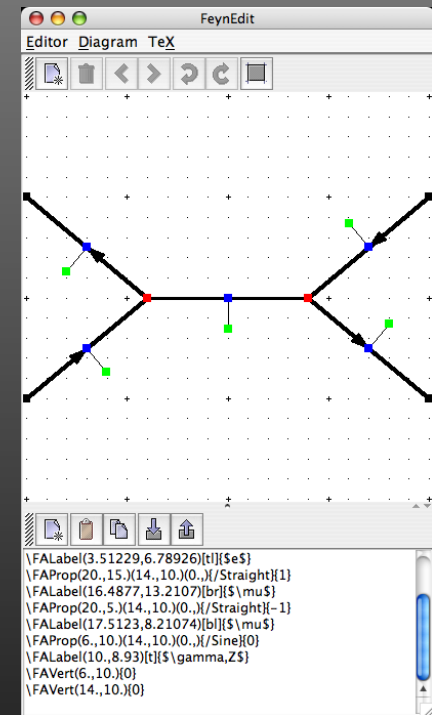
# Editing Feynman Diagrams

The elements of the diagram are easy to recognize and it is straightforward to **make changes e.g. to the label text** using any text editor.

It is less straightforward, however, to alter the geometry of the diagram, i.e. to **move vertices and propagators**.

The **FeynEdit** tool lets the user:

- copy-and-paste the  $\text{\LaTeX}$  code into the lower panel of the editor,
- visualize the diagram,
- modify it using the mouse, and finally
- copy-and-paste it back into the text.



# Excursion: Programming Own Diagram Filters

Or, What if FeynArts' selection functions are not enough.

Observe the structure of inserted topologies:

```
TopologyList[___][t1, t2, ...]  
ti: Topology[___][___] -> Insertions[Generic][g1, g2, ...]  
gi: FeynmanGraph[___][___] -> Insertion[Classes][c1, c2, ...]  
ci: FeynmanGraph[___][___] -> Insertion[Particles][p1, p2, ...]
```

**Example: Select the diagrams with only fermion loops.**

```
FermionLoop[t:TopologyList[___][___]] := FermionLoop/@ t  
FermionLoop[(top:Topology[___][___]) -> ins:Insertions[Generic][___]] :=  
  top -> TestLoops[top]/@ ins  
TestLoops[top_][gi_ -> ci_] := (gi -> ci) /;  
  MatchQ[Cases[top /. List@@ gi,  
    Propagator[Loop[___][v1_, v2_, field_] -> field], {F..}]  
TestLoops[___][___] := Sequence[]
```



# Algebraic Simplification

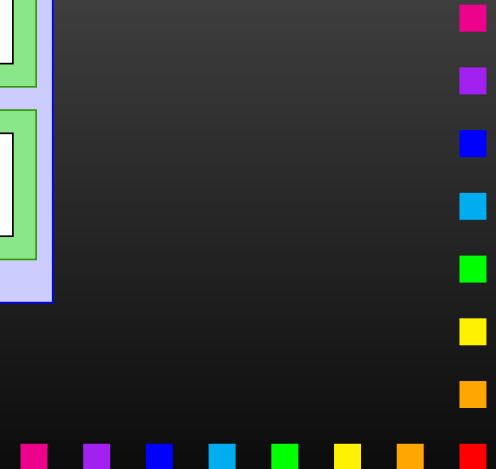
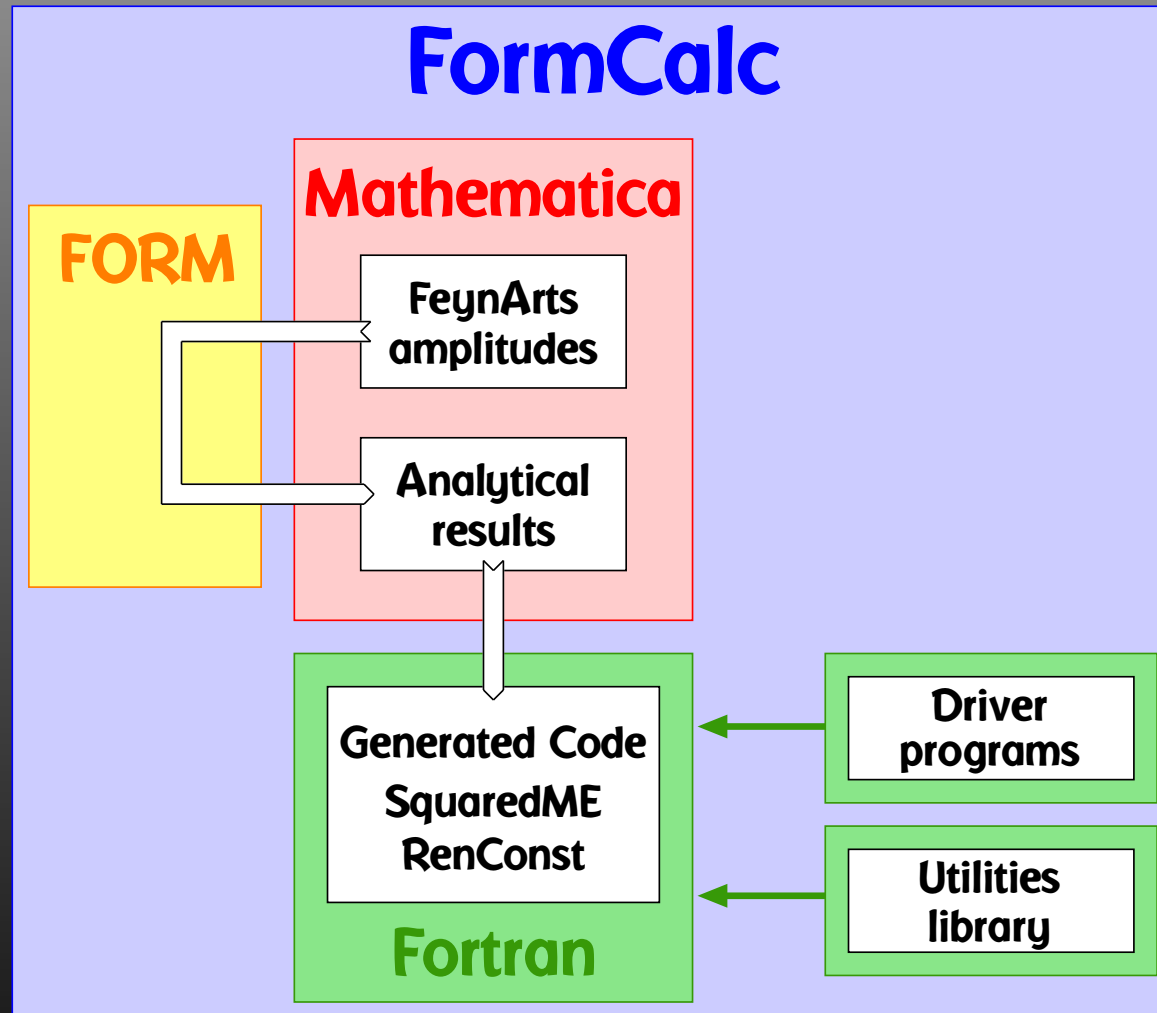
The amplitudes of `CreateFeynAmp` are in **no good shape for direct numerical evaluation.**

A number of steps have to be done analytically:

- **contract indices as far as possible,**
- **evaluate fermion traces,**
- **perform the tensor reduction / separate numerators,**
- **add local terms arising from  $D \cdot$  (divergent integral),**
- **simplify open fermion chains,**
- **simplify and compute the square of  $SU(N)$  structures,**
- **“compactify” the results as much as possible.**



# FormCalc Internals



# FormCalc Output

A typical term in the output looks like

$$\begin{aligned} & C0i[cc12, MW2, MW2, S, MW2, MZ2, MW2] * \\ & ( -4 \text{ Alfa2 } MW2 \text{ CW2/SW2 } S \text{ AbbSum16 } + \\ & 32 \text{ Alfa2 } \text{ CW2/SW2 } S^2 \text{ AbbSum28 } + \\ & 4 \text{ Alfa2 } \text{ CW2/SW2 } S^2 \text{ AbbSum30 } - \\ & 8 \text{ Alfa2 } \text{ CW2/SW2 } S^2 \text{ AbbSum7 } + \\ & \text{ Alfa2 } \text{ CW2/SW2 } S (T-U) \text{ Abb1 } + \\ & 8 \text{ Alfa2 } \text{ CW2/SW2 } S (T-U) \text{ AbbSum29 } ) \end{aligned}$$

 = loop integral

 = kinematical variables

 = constants

 = automatically introduced abbreviations





# Abbreviations

Outright factorization is usually out of question.  
Abbreviations are necessary to reduce size of expressions.

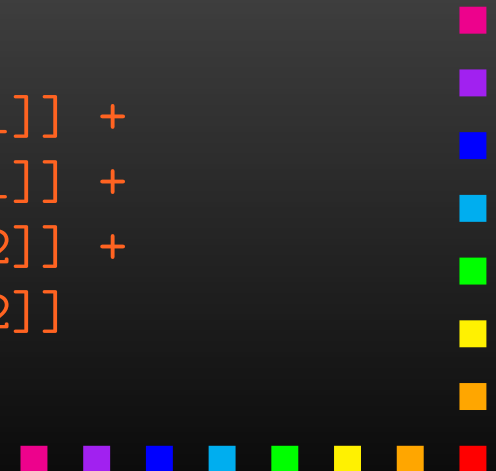
$$\text{AbbSum29} = \text{Abb2} + \text{Abb22} + \text{Abb23} + \text{Abb3}$$

$$\text{Abb22} = \text{Pair1} \text{Pair3} \text{Pair6}$$

$$\text{Pair3} = \text{Pair}[e[3], k[1]]$$

The full expression corresponding to **AbbSum29** is

$$\begin{aligned} & \text{Pair}[e[1], e[2]] \text{Pair}[e[3], k[1]] \text{Pair}[e[4], k[1]] + \\ & \text{Pair}[e[1], e[2]] \text{Pair}[e[3], k[2]] \text{Pair}[e[4], k[1]] + \\ & \text{Pair}[e[1], e[2]] \text{Pair}[e[3], k[1]] \text{Pair}[e[4], k[2]] + \\ & \text{Pair}[e[1], e[2]] \text{Pair}[e[3], k[2]] \text{Pair}[e[4], k[2]] \end{aligned}$$



# Categories of Abbreviations

- Abbreviations are **recursively defined** in several levels.
- When generating code, FormCalc introduces another set of abbreviations for the **loop integrals**.

In general, the **abbreviations are thus costly in CPU time**. It is key to a decent performance that the abbreviations are separated into different **Categories**:

- **Abbreviations that depend on the helicities,**
- **Abbreviations that depend on angular variables,**
- **Abbreviations that depend only on  $\sqrt{s}$ .**

Correct execution of the categories guarantees that **almost no redundant evaluations** are made and makes the generated code essentially as fast as hand-tuned code.



## Excursion: FORM 4 Features in FormCalc

FORM is able to handle **very large expressions**. To produce **(pre-)simplified expressions**, however, terms have to be **wrapped in functions**, to avoid immediate expansion:

$$\begin{aligned} a*(b + c) &\rightarrow a*b + a*c \\ a*f(b + c) &\rightarrow a*f(b + c) \end{aligned}$$

The **number of terms in a function is rather limited in FORM**.

Idea: replace subexpressions by symbols (new FORM 4 feature) once final.

- Prevents expansion, preserves (pre-)simplified structure.
- Introduced symbols are largely inert in further operations.
- Returned (sub)expressions small enough to use fairly aggressive simplification in Mathematica within reasonable run-time.



## More Abbreviations

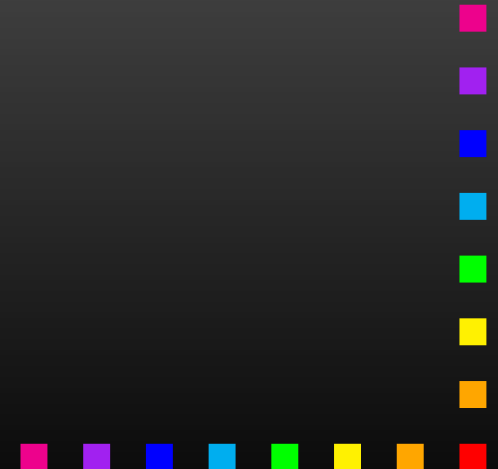
The `Abbreviate` **Function** allows to introduce abbreviations for arbitrary (sub-)expressions and extends the advantage of categorized evaluation.

The subexpressions are **retrieved with** `Subexpr []`.

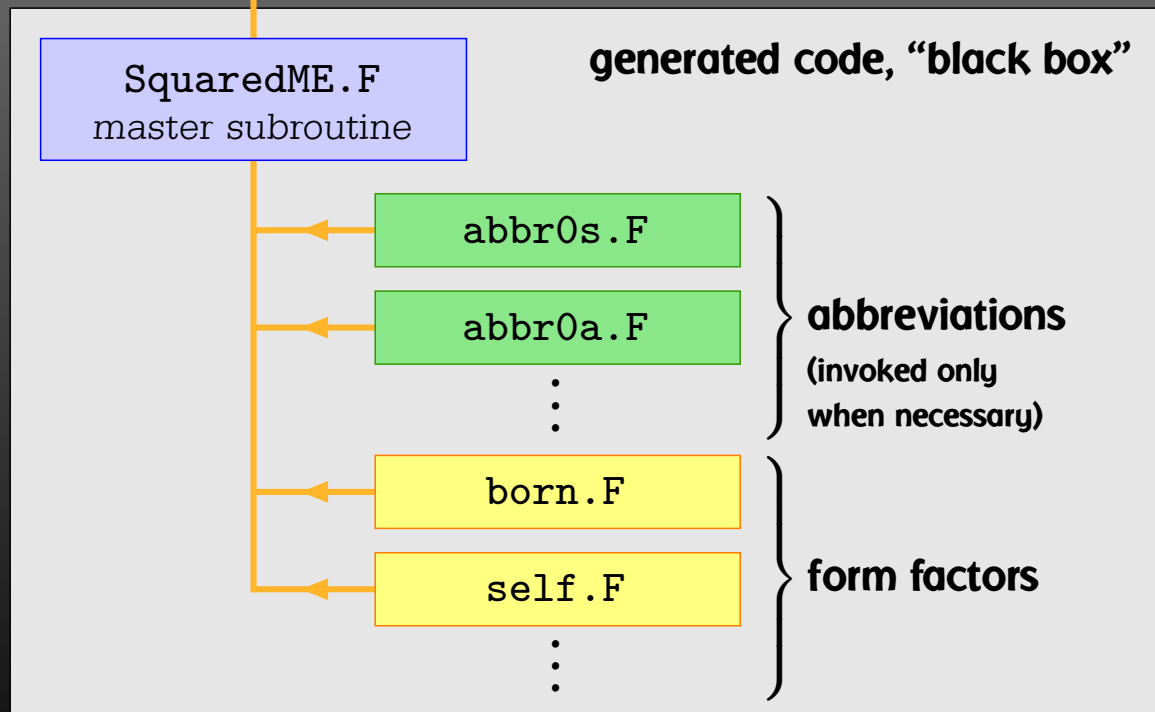
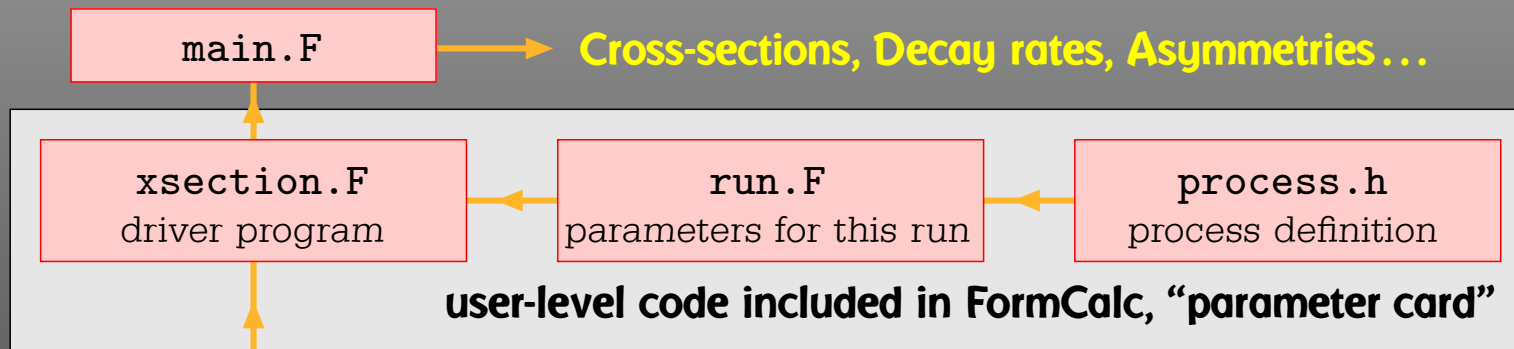
Abbreviations and subexpressions from an earlier `FormCalc` session must be registered before use:

```
RegisterAbbr [abbr]
```

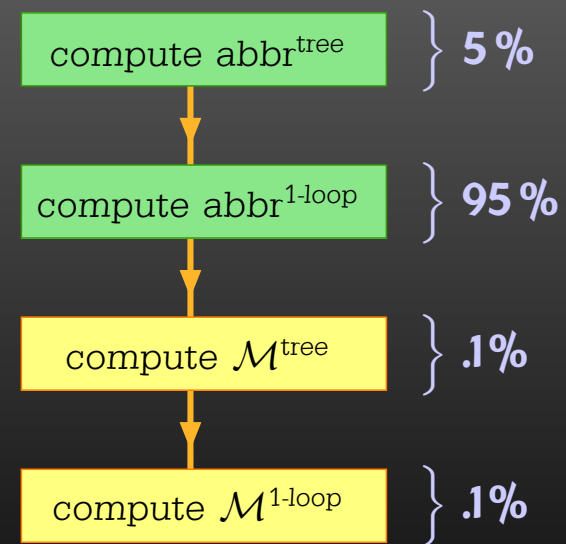
```
RegisterSubexpr [subexpr]
```



# Numerical Evaluation



## CPU-time (rough)



# Features of the Generated Code

- **Extensible:** default code serves (only) as an example. Other 'Frontends' can be supplied, e.g. HadCalc, sofox.
- **Modular:** largely autonomous pieces of code provide
  - kinematics,
  - model initialization,
  - convolution with PDFs.
- **Re-usable:** external program need only call `ProcessIni` (to set up the process) and `ParameterScan` (to set off the calculation).
- **Interactive:** Mathematica interface provides Mathematica function for cross-section/decay rate.
- **Parallel:** built-in parallelization for helicity loop, parameter scans.



# External Fermion Lines

An amplitude containing **external fermions** has the form

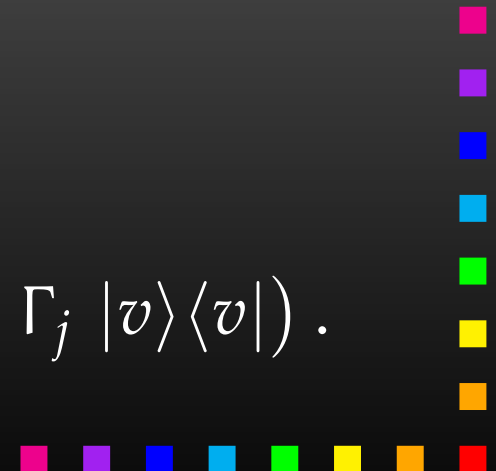
$$\mathcal{M} = \sum_{i=1}^{n_F} c_i F_i \quad \text{where} \quad F_i = \text{(Product of)} \langle u | \Gamma_i | v \rangle .$$

$n_F$  = number of fermionic structures.

Textbook procedure: **Trace Technique**

$$|\mathcal{M}|^2 = \sum_{i,j=1}^{n_F} c_i^* c_j F_i^* F_j$$

**where**  $F_i^* F_j = \langle v | \bar{\Gamma}_i | u \rangle \langle u | \Gamma_j | v \rangle = \text{Tr}(\bar{\Gamma}_i | u \rangle \langle u | \Gamma_j | v \rangle \langle v |)$  .



# Problems with the Trace Technique

**PRO:** Trace technique is independent of any representation.

**CON:** For  $n_F$   $F_i$ 's there are  $n_F^2$   $F_i^* F_j$ 's.

Things get worse the more vectors are in the game:  
multi-particle final states, polarization effects . . .

Essentially  $n_F \sim (\# \text{ of vectors})!$  because all  
combinations of vectors can appear in the  $\Gamma_i$ .

**Solution:** Use Weyl-van der Waerden spinor formalism to  
compute the  $F_i$ 's directly.





# Fermion Chains

FormCalc uses Dirac (4-component) spinors in most of the algebra (extension to  $D$  dim more obvious).

Move to 2-comp. Weyl spinors for the numerical evaluation:

$$\langle u|_4 \equiv (\langle u_+|_2, \langle u_-|_2), \quad |v\rangle_4 \equiv \begin{pmatrix} |v_- \rangle_2 \\ |v_+ \rangle_2 \end{pmatrix}.$$

Every **chiral Dirac chain** maps onto a single Weyl chain:

$$\langle u| \mathbb{P}_L \gamma_\mu \gamma_\nu \cdots |v\rangle_4 = \langle u_-| \bar{\sigma}_\mu \sigma_\nu \cdots |v_\pm\rangle_2,$$

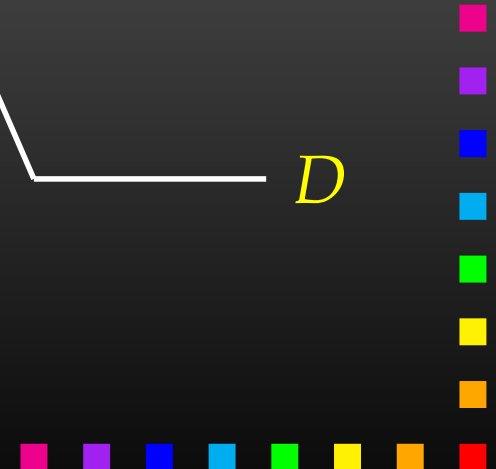
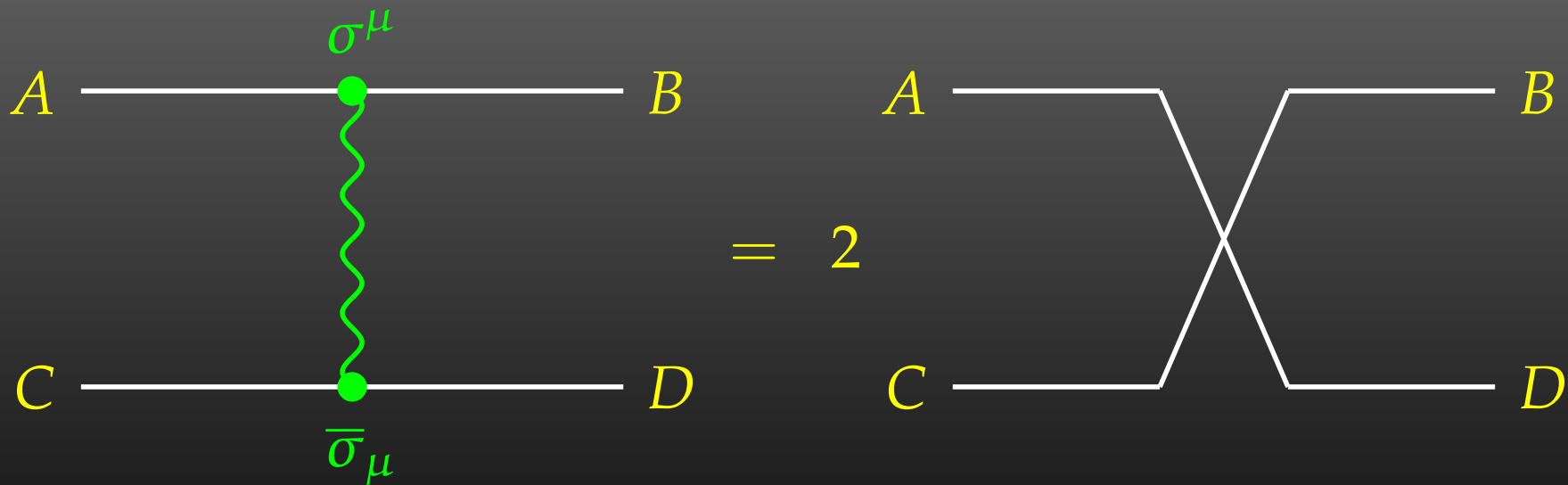
$$\langle u| \mathbb{P}_R \gamma_\mu \gamma_\nu \cdots |v\rangle_4 = \langle u_+| \sigma_\mu \bar{\sigma}_\nu \cdots |v_\mp\rangle_2.$$

**FORM-like notation:**  $\langle u| \sigma_\mu \bar{\sigma}_\nu \sigma_\rho |v\rangle k_1^\mu \varepsilon_2^\nu k_3^\rho \equiv \langle u| k_1 \bar{\varepsilon}_2 k_3 |v\rangle.$

# Fierz Identities

With the Fierz identities for sigma matrices it is possible to **remove all Lorentz contractions** between sigma chains, e.g.

$$\langle A | \sigma_\mu | B \rangle \langle C | \bar{\sigma}^\mu | D \rangle = 2 \langle A | D \rangle \langle C | B \rangle$$



# Implementation

- **Objects:**  $|u_{\pm}\rangle \sim \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}, \quad (\sigma \cdot k) \sim \begin{pmatrix} a & b \\ c & d \end{pmatrix}$

- **Elementary Operations:**

$$\langle u | v \rangle \sim (u_1 \ u_2) \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \quad \text{SxS}$$

$$(\vec{\sigma} \cdot k) |v\rangle \sim \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \quad \text{VxS, BxS}$$

Could fold elementary operations, but faster with single inlined function call:

$$\langle u | \sigma_{\mu} \bar{\sigma}_{\nu} \sigma_{\rho} |v\rangle k_1^{\mu} k_2^{\nu} k_3^{\rho} = \langle u | k_1 \bar{k}_2 k_3 |v\rangle$$

**old** = SxS(u, VxS(k<sub>1</sub>, BxS(k<sub>2</sub>, VxS(k<sub>3</sub>, v))))

**new** = ChainV3(u, k<sub>1</sub>, k<sub>2</sub>, k<sub>3</sub>, v)

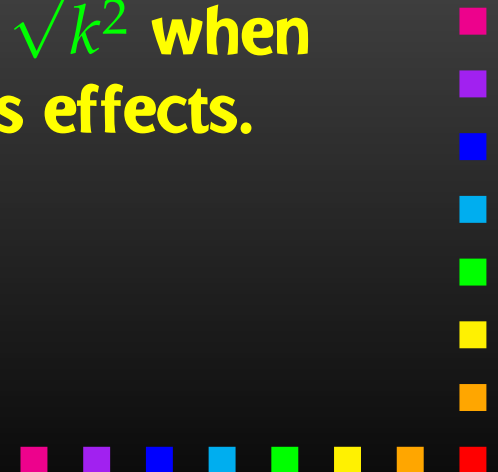
## More Freebies

- Polarization does not 'cost' extra:  
= Get spin physics for free.
- Better numerical stability because components of  $k^\mu$  are arranged as 'small' and 'large' matrix entries, viz.

$$\sigma_\mu k^\mu = \begin{pmatrix} k_0 + k_3 & k_1 - ik_2 \\ k_1 + ik_2 & k_0 - k_3 \end{pmatrix}$$

↓

Large cancellations of the form  $\sqrt{k^2 + m^2} - \sqrt{k^2}$  when  $m \ll k$  are avoided: better precision for mass effects.



# Mathematica Interface

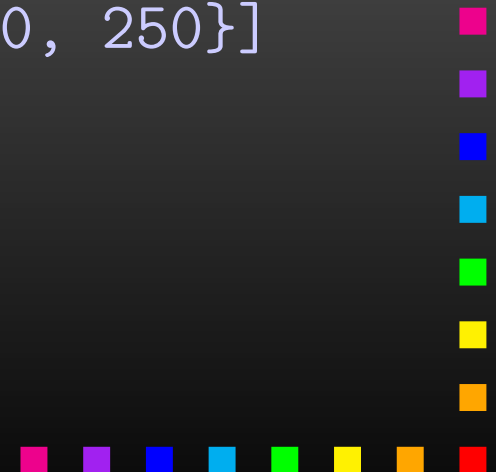
The **Mathematica Interface** turns the generated **stand-alone Fortran code** into a **Mathematica function for evaluating the cross-section or decay rate** as a function of user-selected model parameters.

The benefits of such a function are obvious, as the whole instrumentarium of Mathematica commands can be applied to them. Just think of

```
FindMinimum[sigma[TB, MA0], {{TB, 5}, {MA0, 250}}]
```

```
ContourPlot[sigma[TB, MA0], {TB, 5}, {MA0, 250}]
```

```
...
```



# Mathematica Interface - Input

The changes to the code are minimal.

Example line in `run.F` for Stand-alone Fortran code:

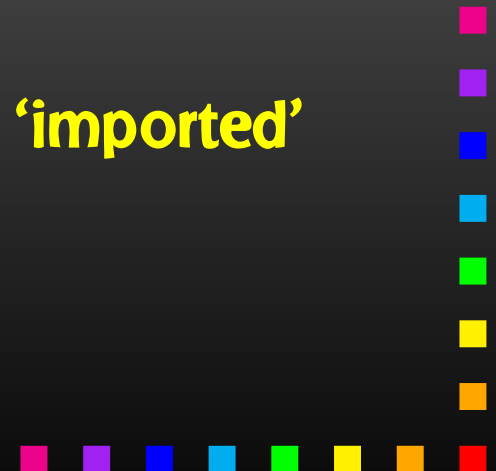
```
#define LOOP1 do 1 TB = 5, 50, 5
```

Change for the Mathematica Interface:

```
#define LOOP1 call MmaGetReal(TB)
```

The variable `TB` is **'imported' from Mathematica** now, i.e. the cross-section function in Mathematica becomes a function of `TB` hereby.

The user has **full control over which variables are 'imported' from Mathematica and which are set in Fortran.**



## Mathematica Interface - Output

Similar to the `MmaGetReal` invocations, the Fortran program can also **'export' variables to Mathematica**.

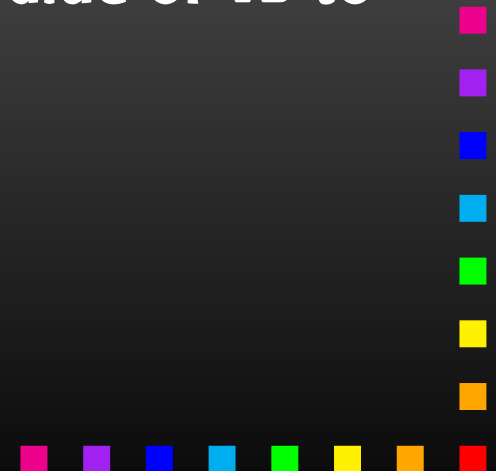
For example, the line that prints a parameter in the stand-alone code is

```
#define PRINT1 SHOW "TB", TB
```

becomes

```
#define PRINT1 call MmaPutReal("TB", TB)
```

for the **Mathematica Interface** and transmits the value of `TB` to **Mathematica**.



## Mathematica Interface - Usage

Once the changes to `run.F` are made, the program `run` is compiled as usual:

```
./configure  
make
```

It is then loaded in Mathematica with

```
Install["run"]
```

Now a Mathematica function of the same name, `run`, is available. There are two ways of invoking it:

**Compute a differential cross-section at  $\sqrt{s} = \text{sqrtS}$ :**

```
run[sqrtS, arg1, arg2, ...]
```

**Compute a total cross-section for**

**$\text{sqrtSfrom} \leq \sqrt{s} \leq \text{sqrtSto}$ :**

```
run[{sqrtSfrom, sqrtSto}, arg1, arg2, ...]
```



# Mathematica Interface - Data Retrieval

The output of the function `run` is an integer which indicates how many records have been transferred. For example:

```
Para[1] = {TB -> 5., MA0 -> 250.}
Data[1] = {DataRow[{500.}, {0.0539684, 0.}, {2.30801 10^-21, 0.}],
          DataRow[{510.}, {0.0515943, 0.}, {4.50803 10^-22, 0.}]}
```

`Para` contains the parameters exported from the Fortran code.

`Data` contains:

- the independent variables,  
here e.g.  $\{500.\} = \{\sqrt{s}\}$ ,
- the cross-sections,  
here e.g.  $\{0.0539684, 0.\} = \{\sigma_{\text{tot}}^{\text{tree}}, \sigma_{\text{tot}}^{\text{1-loop}}\}$ , and
- the integration errors,  
here e.g.  $\{2.30801 \cdot 10^{-21}, 0.\} = \{\Delta\sigma_{\text{tot}}^{\text{tree}}, \Delta\sigma_{\text{tot}}^{\text{1-loop}}\}$ .



# The Model Files

One has to set up, once and for all, a

- **Generic Model File** (seldomly changed) containing the generic part of the couplings,

**Example: the FFS coupling**

$$C(F, F, S) = G_L \mathbb{P}_L + G_R \mathbb{P}_R = \vec{G} \cdot \begin{pmatrix} \mathbb{P}_L \\ \mathbb{P}_R \end{pmatrix}$$

```
AnalyticalCoupling[s1 F[j1, p1], s2 F[j2, p2], s3 S[j3, p3]]  
== G[1][s1 F[j1], s2 F[j2], s3 S[j3]] .  
  { NonCommutative[ ChiralityProjector[-1] ],  
    NonCommutative[ ChiralityProjector[+1] ] }
```



# The Model Files

One has to set up, once and for all, a

- **Classes Model File** (for each model)  
declaring the particles and the allowed couplings

**Example: the  $\bar{\ell}_i \nu_j G$  coupling in the Standard Model**

$$\vec{G}(\bar{\ell}_i, \nu_j, G) = \begin{pmatrix} G_- \\ G_+ \end{pmatrix} = \begin{pmatrix} -\frac{i e m_{\ell,i}}{\sqrt{2} \sin \theta_w M_W} \delta_{ij} \\ 0 \end{pmatrix}$$

```
C[ -F[2,{i}], F[1,{j}], S[3] ]  
== { {-I EL Mass[F[2,{i}]]/(Sqrt[2] SW MW) IndexDelta[i, j]},  
      {0} }
```

# Current Status of Model Files

Model Files presently available for FeynArts:

- **SM [w/QCD], normal and background-field version.**  
**All one-loop counter terms included.**
- **MSSM [w/QCD].**  
**Counter terms now available.**
- **Two-Higgs-Doublet Model.**  
**Counter terms not included yet.**
- **ModelMaker utility generates Model Files from the Lagrangian.**
- **“3rd-party packages” FeynRules and LanHEP generate Model Files for FeynArts and others.**
- **SARAH package derives SUSY Models.**



# Partial (Add-On) Model Files

FeynArts distinguishes

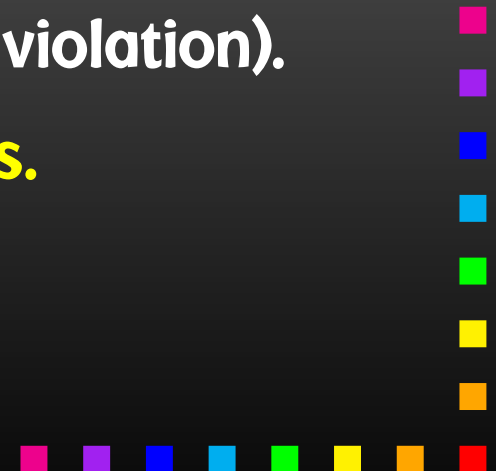
- **Basic Model Files** and
- **Partial (Add-On) Model Files.**

**Basic Model Files, e.g. SM.mod, MSSM.mod, can be modified by Add-On Model Files. For example,**

```
InsertFields[... , Model -> {"MSSMQCD", "FV"}]
```

**This loads the Basic Model File MSSMQCD.mod and modifies it through the Add-On FV.mod (non-minimal flavour violation).**

**Model files can thus be built up from several parts.**



# Tweaking Model Files

Or, How to efficiently make changes in an existing model file.

Bad: Copy the model file, modify the copy. – Why?

- It is typically not very transparent what has changed.
- If the original model file changes (e.g. bug fixes), these do not automatically propagate into the derivative model file.

Better: Create a new model file which reads the old one and modifies the particles and coupling tables.

- `M$ClassesDescription` = list of particle definitions,
- `M$CouplingMatrices` = list of couplings.



# Tweaking Model Files

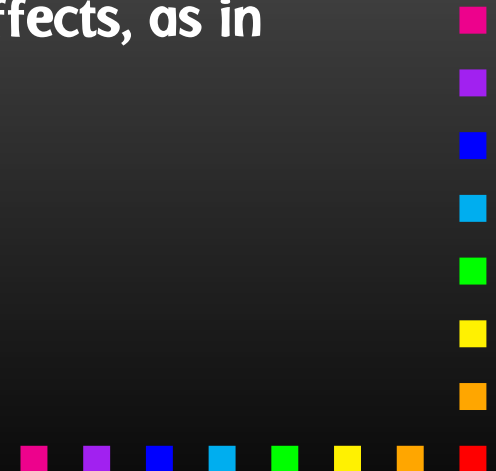
**Example:** Introduce **enhancement factors** for the  $b-\bar{b}-h_0$  and  $b-\bar{b}-H_0$  Yukawa couplings in the MSSM.

```
EnhCoup[ (lhs:C[F[4,{g_,_}], -F[4,_], S[h:1|2]]) == rhs_ ] :=  
  lhs == Hff[h,g] rhs  
EnhCoup[other_] = other  
  
M$CouplingMatrices = EnhCoup/@ M$CouplingMatrices
```

**To see the effect, make a printout with the WriteTeXFile utility of FeynArts.**

**The `Hff[h,g]` can be defined to include e.g. resummation effects, as in**

```
double precision Hff(2,3)  
data Hff /6*1/  
Hff(1,3) = 1 - CA/(SA*TB)*Delta_b  
Hff(2,3) = 1 + SA/(CA*TB)*Delta_b
```

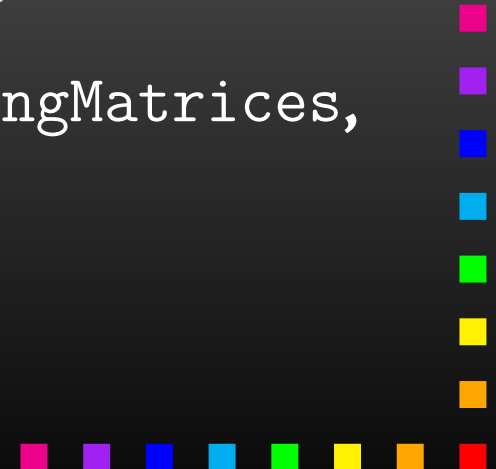


# Linear Combinations of Fields

FeynArts can automatically linear-combine fields, i.e. one can specify the **couplings in terms of gauge rather than mass eigenstates**. For example:

```
M$ClassesDescription = { ...,  
  F[11] = { ...,  
    Indices -> {Index[Neutralino]},  
    Mixture -> ZNeu[Index[Neutralino],1] F[111] +  
              ZNeu[Index[Neutralino],2] F[112] +  
              ZNeu[Index[Neutralino],3] F[113] +  
              ZNeu[Index[Neutralino],4] F[114]} }
```

**Since F[111]...F[114] are not listed in M\$CouplingMatrices, they drop out of the model completely.**





# Linear Combinations of Fields

**Higher-order mixings** can be added, too:

```
M$ClassesDescription = { ...,  
  S[1] = {...},  
  S[2] = {...},  
  S[10] == {...,  
    Indices -> {Index[Higgs]},  
    Mixture -> UHiggs[Index[Higgs],1] S[1] +  
              UHiggs[Index[Higgs],2] S[2],  
    InsertOnly -> {External, Internal}} }
```

**This time,  $S[10]$  and  $S[1], S[2]$  appear in the coupling list (including all mixing couplings) because all three are listed in `M$CouplingMatrices`.**

**Due to the `InsertOnly`,  $S[10]$  is inserted only on tree-level parts of the diagram, not in loops.**



# Not the Cross-Section

Or, How to get things the Standard Setup won't give you.

Example: extract the Wilson coefficients for  $b \rightarrow s\gamma$ .

```
tops = CreateTopologies[1, 1 -> 2]
ins = InsertFields[tops, F[4,{3}] -> {F[4,{2}], V[1]}]
vert = CalcFeynAmp[CreateFeynAmp[ins], FermionChains -> Chiral]

mat[p_Plus] := mat/@ p

mat[r_. DiracChain[s2_Spinor, om_, mu_, s1:Spinor[p1_, m1_, _]]] :=
  I/(2 m1) mat[r DiracChain[sigmunu[om]]] +
  2/m1 r Pair[mu, p1] DiracChain[s2, om, s1]

mat[r_. DiracChain[sigmunu[om_]], SUNT[Col1, Col2]] :=
  r 07[om]/(EL MB/(16 Pi^2))

mat[r_. DiracChain[sigmunu[om_]], SUNT[Glu1, Col2, Col1]] :=
  r 08[om]/(GS MB/(16 Pi^2))

coeff = Plus@@ vert //. abbr /. Mat -> mat

c7 = Coefficient[coeff, 07[6]]
c8 = Coefficient[coeff, 08[6]]
```

# Not the Cross-Section

Using FormCalc's output functions it is also pretty straightforward to **generate your own Fortran code:**

```
file = OpenFortran["bsgamma.F"]

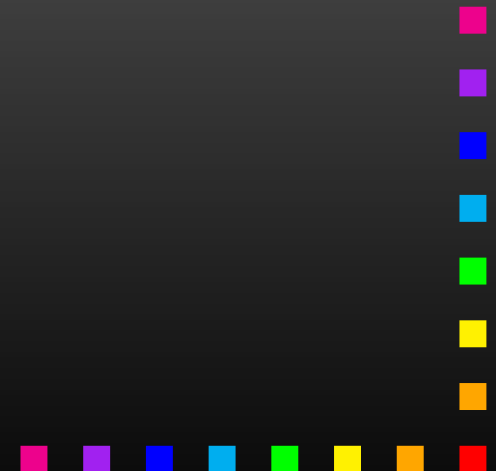
WriteString[file,
  SubroutineDecl["bsgamma(C7,C8)"] <>
  "\tdouble complex C7, C8\n" <>
  "#include \"looptools.h\"\n"]

WriteExpr[file, {C7 -> c7, C8 -> c8}]

WriteString[file, "\tend\n"]

Close[file]
```

**More details in [hep-ph/0607049](https://arxiv.org/abs/hep-ph/0607049).**



# Aiding Operator Matching

As numerical calculations are done mostly using Weyl-spinor chains, there has been a paradigm shift for **Dirac chains** to make them **better suited for analytical purposes**.

- The **Fierz identities rearrange fermion chains** by switching spinors, e.g.

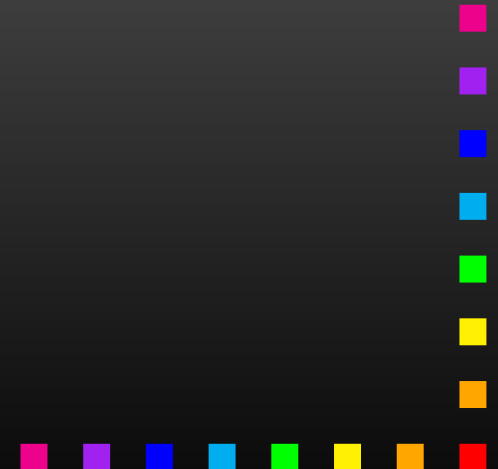
$$\langle 1 | \Gamma_i | 2 \rangle \langle 3 | \Gamma_j | 4 \rangle = \sum c_{kl} \langle 1 | \Gamma_k | 4 \rangle \langle 3 | \Gamma_l | 2 \rangle$$

This is necessary to extract certain predefined structures from the amplitude, most notably **Wilson coefficients**. The **FermionOrder option** of CalcFeynAmp implements **Fierz methods** for Dirac chains including the **Colour** method which brings the spinors into the same order as the external colour indices.



# Aiding Operator Matching

- The `Evanescent option` tracks operators before and after Fierzing for better control of  $\varepsilon$ -dimensional terms.
- The `Antisymmetrize option` allows the choice of **completely antisymmetrized Dirac chains**, i.e.  
 $\text{DiracChain}[-1, \mu, \nu] = \sigma_{\mu\nu}$ .



# Summary and Outlook

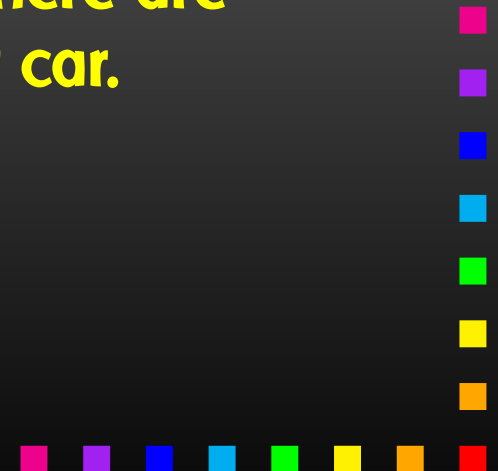
- **Serious perturbative calculations these days can generally no longer be done by hand:**
  - Required accuracy, Models with many particles, ...
- **Hybrid programming techniques are necessary:**
  - Computer algebra is an indispensable tool because many manipulations must be done symbolically.
  - Fast number crunching can only be achieved in a compiled language.
- **Software engineering and further development of the existing packages is a must:**
  - As we move on to ever more complex computations (more loops, more legs), the computer programs must become more “intelligent,” i.e. must learn all possible tricks to still be able to handle the expressions.



# Finally

Using FeynArts and FormCalc is a lot like driving a car:

- You have to decide where to go (this is often the hardest decision).
- You have to turn the ignition key, work gas and brakes, and steer.
- But you don't have to know, say, which valve has to open at which time to keep the motor running.
- On the other hand, you can only go where there are roads. You can't climb a mountain with your car.



## Exercise

Devise a Model File with a fermion, a vector, and a scalar,  
where

$$C(F, F, V) = V_- \gamma_\mu \omega_- + V_+ \gamma_\mu \omega_+$$

$$C(F, F, S) = S_- \omega_- + S_+ \omega_+$$

<http://wwwth.mpp.mpg.de/members/hahn/fafc.pdf>

<http://wwwth.mpp.mpg.de/members/hahn/fafc.tar.gz>

